L7- Collection Objects

January 28, 2020

1 Collection Objects

Java has two major types of collection objects:

- **`ArrayList`**: size is variable
- **`Array`**: size is fixed

Many applications involve collections of objects, such as libary catalogs, student records. The number of items to be stored may vary. Both collection object build on the concept of **abstraction** from the last chapter.

1.1 Music Organizer Example

[2]:
```
import java.util.ArrayList;

/**
 * A class to hold details of audio files.
 *
 * @author David J. Barnes and Michael Kölling
 * @version 2016.02.29
 */
public class MusicOrganizer
{
    // An ArrayList for storing the file names of music files.
    private ArrayList<String> files;

    /**
     * Create a MusicOrganizer
     */
    public MusicOrganizer()
    {
        files = new ArrayList<>();
    }

    /**
     * Add a file to the collection.
     * @param filename The file to be added.
     */
```

```
    public void addFile(String filename)
    {
        files.add(filename);
    }

    /**
     * Return the number of files in the collection.
     * @return The number of files in the collection.
     */
    public int getNumberOfFiles()
    {
        return files.size();
    }

    /**
     * List a file from the collection.
     * @param index The index of the file to be listed.
     */
    public void listFile(int index)
    {
        if(index >= 0 && index < files.size()) {
            String filename = files.get(index);
            System.out.println(filename);
        }
    }

    /**
     * Remove a file from the collection.
     * @param index The index of the file to be removed.
     */
    public void removeFile(int index)
    {
        if(index >= 0 && index < files.size()) {
            files.remove(index);
        }
    }
}
```

- we have an `ArrayList` of `String`s to store the filenames. This is initialized to empty in the constructor.
- the type in the `ArrayList` is *not* specified in the constructor. This is known as *type inference*.
- `getNumberOfFiles()` just uses the `ArrayList.size()` method
  - this is called *delegation*

1.2 Class Libraries

Note that we need to import the arraylist type with `Java.util.ArrayList`.

These libraries are collections of useful classes. These are called **packages**. Grouping objects happens a lot; loads of ways to do this in `Java.util`.

1.3 Generic Classes

Collections are generic or paramterized types:

- `ArrayList` implements all list functionality
- The parameter tells the collection what we are collecting in it: `<Cat>`s, `<Dogs>`s, `<TicketMachine>`, or whatever.
  - Everything is an object; the contents of the array are already objects.
- If no type is specified, it may be inferred from the type of the variable.

1.3.1 Features of the collection

The `ArrayList`:

- increases its size as necessary
- keeps a count of items, accessed with `ArrayList.size()`
- keeps objects in order
  - removing items from the beginning of the list affects the numbering of items that come after
- doesn't force us to care about how it does any of this
- numbers its indices
  - index starts a 0, goes to n-1 for n items in the list
  - lets us do useful stuff:
    - get the first thing: `ArrayList.get(0)`
    - get the last thing: `ArrayList.get( ArrayList.size() - 1 )`
    - and others…

1.3.2 Retrieving from the collection

- `ArrayList.get(index)`: return the item in the `ArrayList` at `index`
- should generally be wrapped in enough logic to ensure that the index is valid
- see `MusicOrganizer.listFile()`

1.4 Iterating over collections

Sometimes, with a collection, we would like to do stuff to them, so it would be neat if we could iterate over them.

Java has several kinds of loop. We will start with the **for-each** loop.

1.4.1 For-each

- do an action exactly once for each object in the collection
- Using each *element* in *collection* in order, *do_something*

[ ]:
```
for (ElementType element: collection)
{
    do_something();
}
```

[ ]:
```
public void listAllFiles()
{
    for ( String filename: files )
    {
        System.out.println( filename );
    }
}
```

Statements can be nested; we could put an `if/else` inside the loop:

[ ]:
```
public void findFiles(String searchQuery)
{
    for ( String filename : files )
    {
        if ( filename.contains(searchQuery) )
        {
            System.out.println(filename);
        }
    }
}
```

These are:

- easy
- naturally terminating (a *definite* loop)
- **cannot be stopped partway**
- **cannot change the collection**

1.4.2 While loop

- more flexible than the for loop

- uses a Boolean condition to decide whether or not to continue iterating
- an *indefinite loop*
  - have no idea how many places to look for your keys
  - we will stop when some state is true
  - might get an *infinite loop* through errors or just the nature of the task

[ ]:
```
// while syntax

while (condition)
{
    do_something()
}
```

[ ]:
```
// a for loop as a while loop
// can access the index

public void noodles()
{
    int index = 0;
    while (index < somelist.size()))
    {
        do_something();
        index++;
    }
}
```

Reasons not to do this:

- have to come up with our own stopping conditions
- need to specify an index
- need to debug all that garbage

But if you need the index, need to process part of a collection, it's the only game in town. It need not be used with a collection at all.

1.5 Searching

To search:

- inherently indefinite
- must handle success and failure
  - success: found a thing
  - failure: nowhere left to look
- `while` loop syntax means we must specify the conditions to *continue*

  - `searching == true && index < list.size()`
  - if we find a thing:
    * `searching = false`
- alternately:
  - `found == false && index < list.size()`
  - `found = true` if we find a thing
- if we do all that and reach the end of the loop, either we found the thing at some index or we got to the end of the collection without finding it

Note that a while loop can be executed **zero** times if the condition is never true.

[ ]: